Fault-Tolerant Relay Deployment for k Node-Disjoint Paths in Wireless Sensor Networks

Lanny Sitanayah
Mobile & Internet Systems Laboratory
Department of Computer Science
University College Cork, Ireland
Email: ls3@cs.ucc.ie

Kenneth N. Brown
Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
Email: k.brown@cs.ucc.ie

Cormac J. Sreenan
Mobile & Internet Systems Laboratory
Department of Computer Science
University College Cork, Ireland
Email: cjs@cs.ucc.ie

***Abstract*—Ensuring that wireless sensor networks (WSNs) are robust to failures requires that the physical network topology will offer alternative routes to the sinks. This requires sensor network deployments to be planned with an objective of ensuring some measure of robustness in the topology, so that when failures occur that routing protocols can continue to offer reliable delivery. Our contribution is a solution that enables fault-tolerant WSN deployment planning by judicious use of a minimum number of additional relay nodes. A WSN is robust if at least one route to a sink is available for each remaining sensor node after the failure of up to $k-1$ nodes. In this paper, we define the problem for increasing WSN reliability by deploying a number of additional relay nodes to ensure that each sensor node in the initial design has k node-disjoint paths to the sinks. We present GRASP-ARP, a centralised offline algorithm to be run during the initial topology design to solve this problem. We have implemented this algorithm and demonstrated in simulation that it improves the efficiency of relay node placement for k node-disjoint paths compared to the most closely related published algorithms.**

***Index Terms*—wireless sensor networks; network deployment planning; relay placement; node-disjoint paths.**

I. Introduction

In wireless sensor networks (WSNs), sensor nodes transmit their data wirelessly over a multi-hop network to sink nodes, where data is either processed or transmitted on through a high-speed connection. These networks are subject to failures: the wireless devices are often unreliable, they have limited battery life, transmissions may be blocked by changes in the environment, and the devices may be damaged, e.g. by weather, wildlife or human intervention. For dealing with failures, techniques for reliable routing in WSNs have been proposed and are well-understood, but to be effective these depend on a physical network topology that ensures alternative routes to the sink are available. This requires sensor network deployment to be planned with an objective of ensuring some measure of robustness in the topology, so when failures occur that routing protocols can continue to offer reliable delivery. Our contribution is a solution that enables fault-tolerant WSN deployment planning by judicious use of additional relays.

One key objective in the topology design of a WSN is to ensure some measure of robustness. In particular, one standard criterion is to ensure routes to the sink are available for all remaining sensor nodes after the failure of up to $k-1$ nodes. This can be achieved by ensuring that every node in the initial design has k node-disjoint paths to the sinks: i.e. at least k paths that share no intermediate nodes. Node-disjoint paths are required to provide multi-path routing capability for some protocols [1]. Since there are sometimes data latency requirements, there may be a limit to the path length from sensor to sink. To ensure that the sensors have sufficient paths, it may be necessary to add a number of additional relays, which do not sense, but only forward data from other nodes. Installing additional relays comes at a cost that includes not just the hardware purchase but more significantly the installation and ongoing maintenance, thus motivating solutions that minimise the number of additional relays.

The problem we address in this paper is that of finding a minimal set of relays which ensures k node-disjoint paths for each sensor. We define the single-tiered, constrained partial fault-tolerant relay placement problem for k node-disjoint paths with length constraints for WSNs with data sinks, which we call the additional relay problem (ARP). We make no assumptions on the geographical or physical properties of the area in which the WSN is to be deployed, but we assume a limited set of possible locations for the relays, and a fully-specified connectivity graph. We propose GRASP-ARP, a local search algorithm based on GRASP [2], to be run as a centralised offline algorithm during the initial topology design, i.e. prior to network deployment and operation. Our algorithm requires repeated counting of the number of node-disjoint paths for each node, for which we use a dynamic programming procedure. We compare GRASP-ARP against the closest approach from the literature, and we demonstrate empirically that it finds solutions requiring 25% fewer additional relays for small values of k. We also show that it scales better, finding solutions in reasonable time for problems with hundreds of nodes. This represents a quantifiable improvement, making it possible to compute cost-effective WSN designs that offer an assured level of reliable delivery.

The rest of the paper is organised as follows. We survey the related work in Section II, present the additional relay problem and GRASP-ARP in Section III and IV, respectively. Section V presents simulation results and Section VI concludes.

TABLE I
EXISTING RELAY PLACEMENT ALGORITHMS

Algorithms	k	Routing	Deployment Locations	Fault-Tolerance
Hao *et al.* [3]	2	2-tiered	constrained	partial
Bredin *et al.* [4]	≥ 1	1-tiered	unconstrained	full
Pu *et al.* [5]	≥ 1	1-tiered	unconstrained	partial
Kashyap *et al.* [6]	≥ 2	2-tiered	un/constrained	partial
Han *et al.* [7]	≥ 1	1-tiered	unconstrained	full/partial
Zhang *et al.* [8]	2	1/2-tiered	unconstrained	partial
Misra *et al.* [9]	1, 2	1-tiered	constrained	partial

II. BACKGROUND AND RELATED WORK

A WSN can be modeled as a graph $G=(V,E)$, where V is a set of nodes and E is a set of edges. Each edge connects two nodes that are within transmission range of each other[1], and the two nodes are said to be *adjacent*. A *path* of length t between two nodes v and w is a sequence of nodes $v=v_0,v_1,\ldots,v_t=w$, such that v_i and v_{i+1} are adjacent for each i. A path from a node v to a set of nodes W is simply a path from v to any node $w\in W$. Two nodes are *connected* if there is a path between them. A graph is connected if every pair of nodes is connected. A *cutset* is a set $C\subset V$ such that $(V-C, E{\downarrow}_{V-C})$ is disconnected (where $E{\downarrow}_X$ means a set of edges restricted to those that connect nodes in X). A graph is k-*connected* if it has no cutset of size less than k. Two paths P and Q from v to w are *node-disjoint* if they have no nodes in common except for v and w.

The problem of placing relay nodes for increased reliability has long been acknowledged as a significant problem, and we summarise the existing algorithms for WSNs in Table I. In *single-tiered* networks, all nodes may forward packets from other nodes, while in *two-tiered* networks, sensor nodes transmit their own data directly to a cluster head. In *connectivity* problems, the aim is to ensure the network is connected (or $k=1$), while in *survivable* problems, the aim is to ensure k-connectivity for $k>1$. In *unconstrained* problems, the relay nodes can be placed anywhere, while in *constrained* problems, they are limited to a set of candidate locations. *Partial fault-tolerance* requires k-connectivity between every sensor node, while *full fault-tolerance* requires k-connectivity between both sensor and relay nodes. We choose to focus on single-tiered networks as this is most common in the research literature and for published WSN deployments. Furthermore we assume the constrained approach for possible relay locations, which we believe is more reasonable for real-world deployments. Finally we assume partial fault-tolerance, reflecting the fact that relays are for connectivity only and do not have a sensing role.

All of the algorithms cited try to achieve k-connectivity, and are designed for WSNs without designated sinks. Unlike our algorithm, the published algorithms do not place any constraints on the path lengths. The closest problem definition to ours is Misra *et al.*'s [9]. However, it can only establish up to 2-connected networks due to the use of an existing algorithm to compute a low weight bi-connected subgraph. Other work with similar objectives to ours include Bredin *et al.* [4], Pu *et al.* [5] and Han *et al.* [7], although they are for unconstrained deployment locations. Bredin *et al.*'s [4] considers full fault-tolerant relay node placement. This was then modified by Pu *et al.* [5] for partial fault-tolerance after noting that there is no need to ensure multiple paths for the relays. The simulation results in Han *et al.*'s work [7] show that Bredin *et al.*'s [4] is more efficient for partial fault-tolerance, while Han *et al.*'s [7] is more efficient for full fault-tolerance in terms of the number of additional relays. Therefore, among all existing algorithms, we infer that the most relevant one for our work is Pu *et al.*'s [5], except that it assumes unconstrained relay locations.

Offline algorithms to discover k shortest node-disjoint paths in existing networks are well studied in the literature. Torrieri [1] calculates a set of short node-disjoint paths in polynomial time. Bhandari [10] proposes k runs of a modified Dijkstra algorithm, each of which requires $O(|V|^2)$, to find k shortest node-disjoint paths between a source and a sink. There is a close relation between k-connectivity and maximum flow problems. Maximum flow algorithms, such as Ford-Fulkerson [11], are used to find link-disjoint paths [12], and thus need to be extended with a node-splitting technique as used by Bhandari [10]. Link-disjoint paths share no links, but are less robust than node-disjoint paths, and so we restrict our discussion to the latter.

The greedy randomized adaptive search procedure (GRASP) [2] is a metaheuristic intended to capture the good features of pure greedy algorithms and of random construction procedures. It is an iterative process, which consists of two phases: a construction phase and a local search phase. Martins *et al.* [13] use GRASP to solve the Steiner tree problem in graphs (SPG). SPG is similar to the relay placement problem, in that it must select from a set of candidate nodes in order to connect a number of designated terminals, although its aim is to find a minimal spanning tree rather than a forest with node-disjoint paths.

III. ADDITIONAL RELAY PLACEMENT

We consider WSNs where the nodes are partitioned into sinks, sensors and relays, and so in the graph representation $V=S\cup N\cup R$. We define a WSN to be *(k,l)-sink-connected* if and only if for every sensor node $v\in N$, there are k node-disjoint paths from v to S of length $\leq l$.

We can now define the *additional relay placement problem*: given a graph $G=(S\cup N\cup A, E)$, where A is a set of candidate positions for relay nodes, find a minimal subset $R\subseteq A$ such that $H=(S\cup N\cup R, E{\downarrow}_{S\cup N\cup R})$ is (k,l)-sink-connected.

For our algorithm, we introduce some secondary definitions. k_v is the number of node-disjoint paths a sensor currently has. $X\subseteq N$ is the set of sensors with $k_x<k, \forall x\in X$. $Y\subseteq N$ is the set of sensors with $k_y\geq k, \forall y\in Y$, and such that y is on a path of a sensor $x\in X$ to a sink or a node with at least k node-disjoint paths. $Z\subseteq N$ is a set of sensors with $k_z\geq k, \forall z\in Z$, such that z does not appear on a path of a sensor $x\in X$ to a sink or a node with k node-disjoint paths.

[1]For simplicity we assume bi-directional links, but this could be easily relaxed by specifying a more complex connectivity graph.

IV. GRASP-ARP

A. Construction Phase

The first step in any GRASP algorithm is to construct an initial solution. We generate a graph $G' = (V', E')$, where $V' = \{S \cup X \cup Z\}$ and $E' = \{(v, w), v \in X, w \in \{S \cup Z\} \mid \exists srp(v, w)\}$. $srp(v, w)$ denotes the shortest relay path from v to w in the original graph G in terms of the cost c, where all intermediate nodes in the path are candidate relays. The link's cost $c'(v, w) = srp(v, w)$ is associated with each link $(v, w) \in E'$. After that, we find a minimum forest F of G' such that for each $v \in X$, we select $k - k_v$ least cost links to $w \in \{S \cup Z\}$. Then, we replace the links in F by the links in the shortest relay paths in the original graph G and save this set of paths P. We add randomisation to the initial solution by building a restricted candidate list with all links $(v, w) \in E'$ such that $c'(v, w) \leq c'_{min} + \alpha(c'_{max} - c'_{min})$, where $0 \leq \alpha \leq 1$. c'_{min} and c'_{max} denote the least and the largest costs among all unselected links, respectively. Then, $k - k_v$ links are selected at random from the restricted candidate list.

B. Node-based Local Search

The next stage in a GRASP algorithm is to explore the neighbourhood of the initial solution, looking for lower cost solutions. Let R be the set of relays in the current forest F and $R_{used}(P)$ is the number of relays used in the current set of paths P. We explore the neighbourhood of the solution by either adding a new relay $r \in \{A \backslash R\}$ into R or by eliminating a relay $t \in R$ from R. In each iteration, elimination moves are evaluated only if there are no improving insertion moves.

C. Algorithm Description

The pseudocode for GRASP-ARP is given in Algorithm 1. It takes as input the original graph $G = (V, E)$, the set S of sinks, the set A of candidate relays, the set X of sensors with $k_x < k$, the set Z of sensors with $k_z \geq k$ but do not appear on any discovered paths, the restricted candidate list parameter α, and the number of iterations. Graph $G' = (V', E')$ is computed in line 2. The procedure is repeated *max_iterations* times. In each iteration, a greedy randomised solution for a minimum forest F of G' is constructed in line 4. Let R be the set of relays in the current forest F, P be the set of paths, and $R_{used}(P)$ be the number of relays used in P.

The local search starts with the initialisation in line 8. The loop from line 9 to 14 searches for the best insertion move. In line 10, we count the number of node-disjoint paths k_x from each sensor $x \in X$, defined by the insertion of node r into the current set of relay nodes. Let P_{new} be its new set of paths. In line 11, we check if this new solution P_{new} improves the current best solution. Solution updates are made in line 12. When all insertion moves have been evaluated, we check in line 15 if an improving solution has been found and update the solution in line 16. Then, the local search continues. If no improving solution is found from the insertion moves, the elimination moves are evaluated. These moves are similar to the insertion moves, except they are defined by the elimination of node t from the current set of relay nodes.

GRASP-ARP $(G, S, A, X, Z, \alpha, max\_iterations)$

```
1  best_value ← ∞;
2  Compute G' = (V', E') and c'(v, w), ∀(v, w) ∈ E';
3  for i ← 1 to max_iterations do
                                        /* Construction phase */
4      Find F of G' = (V', E') with R and P as the result;
5      do
6          do
7              insertion ← false; elimination ← false;
                                        /* Insertion moves */
8              best_set ← R; best_number ← R_used(P);
9              for all r ∈ {A \ R} do
10                 Count k_x, ∀x ∈ X from F^{+r}, result in P_new;
11                 if R_used(P_new) < best_number then
12                     best_set ← R ∪ {r};
                       best_number ← R_used(P_new);
13                 end if;
14             end for;
15             if best_number < R_used(P) then
16                 R ← R ∪ {r}; F ← F^{+r}; P ← P_new;
                   R_used(P) ← R_used(P_new);
17                 insertion ← true;
18             end if;
19         while insertion;
                                        /* Elimination moves */
20         best_set ← R; best_number ← R_used(P);
21         for all t ∈ R do
22             Count k_x, ∀x ∈ X from F^{-t}, result in P_new;
23             if R_used(P_new) < best_number then
24                 best_set ← R \ {t};
                   best_number ← R_used(P_new);
25             end if;
26         end for;
27         if best_number < R_used(P) then
28             R ← R \ {t}; F ← F^{-t}; P ← P_new;
               R_used(P) ← R_used(P_new);
29             elimination ← true;
30         end if;
31     while elimination;
                                        /* Best solution update */
32     if R_used(P) < best_value then
33         R* ← R; P* ← P; best_value ← R_used(P);
34     end if;
35 end for;
36 return R*, P*;
```

Algorithm 1: GRASP-ARP

If at the end of the local search we found a better solution compared to the best solution, updates are made in line 33. The best set R^* of relay nodes and the best set P^* of paths are returned in line 36.

D. Counting Node-Disjoint Paths

In lines 10 and 22 of Algorithm 1, we count the number of node-disjoint paths from a sensor node to the sinks. We introduce Counting-Paths, a heuristic algorithm that counts the number of node-disjoint paths from each sensor and finds the shortest node-disjoint paths to the sinks. The idea behind Counting-Paths is to use the Ford-Fulkerson [11] maximum flow algorithm, but this only discovers link-disjoint paths. However, by utilising the node-splitting technique as used by Bhandari [10], we can find node-disjoint paths. The time

complexity of the Ford-Fulkerson algorithm for a pair of source and sink is $O(|E|k)$, where k is the number of node-disjoint paths. We refer to [14] for simulation results that validate the efficiency of Counting-Paths compared to [10] and [1].

The dynamic programming implementation of Counting-Paths is motivated by the fact that multi-hop WSNs are often characterised by many-to-one (convergecast) traffic patterns. If the network has n source nodes, running Counting-Paths n times increases the worst time complexity to $O(|V||E|k)$. However, if we do not need to know the complete routes during the deployment process, it is not necessary for us to discover the actual paths, but only the number of node-disjoint paths that satisfies the length restriction and the neighbours that have k node-disjoint paths. This local information is used by nodes to forward their data to the nearest neighbours and the neighbours will have their own decisions to forward them further. Below, we prove the result that justifies our dynamic programming approach.

Lemma 4.1: Let v be a node which has node-disjoint paths to a subset W of k nodes none of which have a cutset of size $<k$. Then v has no cutset of size $<k$.

Proof: Suppose v does have a cutset, C, of size $<k$. A set of size $<k$ can break at most $k-1$ of the paths from v to W. Let $w \in W$ be any of the nodes whose paths from v are not broken by C, and so v is still connected to w. But w must be connect to S, since w has no cutset of size $<k$. Therefore v is still connected to S. Therefore C is not a cutset for v. Contradiction. ∎

As a corollary, if a node v has node-disjoint paths to k nodes, each of which has k node-disjoint paths to the sink, then v must also have k node-disjoint paths to the sink.

For WSNs with multiple sinks, a well-known approach is by adding a *supersink* as an imaginary node that has connection to the original sinks. By doing this, we reduce the problem of multiple sinks to the problem of single sink.

E. Acceleration Scheme

We follow the acceleration scheme for the node-based local search as in [13] by using faster implementation of the insertion moves (line 8-18). The idea is to keep a candidate list with promising insertion moves, which is periodically updated. The candidate list containing the *k_best* improving insertion moves is generated in the first iteration. The list is kept in an increasing order of the associated move values. In each following iteration, instead of reevaluating all insertion moves, we only evaluate moves from nodes in the candidate list. Each time a node is evaluated, it is removed from the list. When the list becomes empty, a new full iteration is performed, all insertion moves are evaluated, and the candidate list is rebuilt.

V. Simulation and Results

All algorithms were evaluated on a custom simulator written in C++. We do not use standard network simulators because we are not evaluating network protocols and operations, but rather the performance of algorithms that are used in planning a network. Our simulation results are based on the mean value of 20 randomly generated network deployments, enough to achieve a 95% confidence interval. The network consists of up to 225 sensor nodes deployed within randomly perturbed grids, where a sensor node is placed in a unit grid of 8m × 8m and the coordinates are slightly perturbed. Candidate relays are also distributed in a grid area, where a candidate occupies a unit grid of 4m × 4m. Both sensor nodes and relay nodes use the same transmission range, i.e. 10 metres.

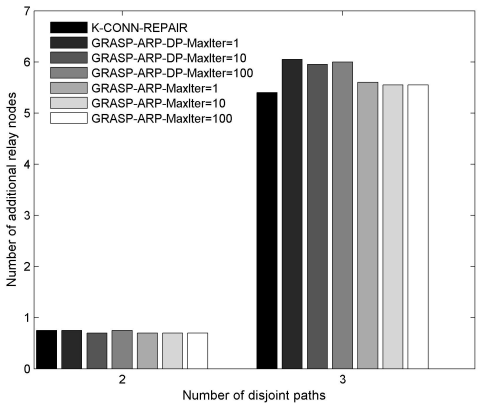

Fig. 1. Number of additional relay nodes needed for the multiple sources - single corner sink in 25-node networks

TABLE II
Relay placement algorithms' run time for the multiple sources - single corner sink in 25-node networks

Algorithms	Run time (sec)	
	$k=2$	$k=3$
K-CONN-REPAIR	8.1711	23.5087
GRASP-ARP-DP-MaxIter=1	1.3539	61.7501
GRASP-ARP-DP-MaxIter=10	3.9977	200.3548
GRASP-ARP-DP-MaxIter=100	26.7742	1573.4410
GRASP-ARP-MaxIter=1	1.3032	91.3281
GRASP-ARP-MaxIter=10	4.5867	244.6632
GRASP-ARP-MaxIter=100	36.8587	1814.2370

We compared GRASP-ARP to the partial k-connectivity repair algorithm (K-CONN-REPAIR for short) proposed by Pu *et al.* [5]. We followed the algorithm detailed in [5] and made two necessary modifications to work in constrained deployment locations, where relay nodes can only be placed in specific candidate locations. The original K-CONN-REPAIR deploys relay nodes along a straight line between two sensor nodes. Therefore, our first modification is to place relay nodes in candidate locations along the shortest relay path between two sensor nodes. When all relay nodes are deployed, our second modification tries to remove relay nodes one by one by still preserving the k-connectivity. We use a maximum network flow based checking algorithm [15], [16] to check the connectivity. In the simulation, we compared the performance of GRASP-ARP against K-CONN-REPAIR in terms of the number of additional relay nodes needed and the run time.

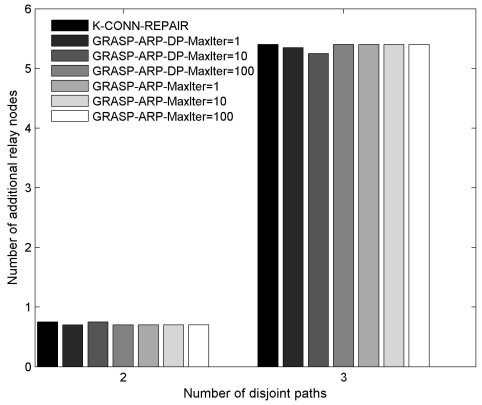

Fig. 2. Number of additional relay nodes needed for the multiple sources - single centre sink in 25-node networks

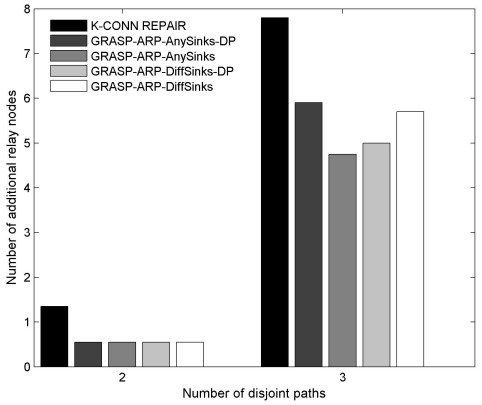

Fig. 3. Number of additional relay nodes needed for the multiple sources - multiple sinks in 49-node networks

TABLE III
RELAY PLACEMENT ALGORITHMS' RUN TIME FOR THE MULTIPLE SOURCES - SINGLE CENTRE SINK IN 25-NODE NETWORKS

Algorithms	**Run time (sec)**	
	$k=2$	$k=3$
K-CONN-REPAIR	8.1711	23.5087
GRASP-ARP-DP-MaxIter=1	1.2009	18.6055
GRASP-ARP-DP-MaxIter=10	3.2695	56.4545
GRASP-ARP-DP-MaxIter=100	29.1741	396.7828
GRASP-ARP-MaxIter=1	1.3203	27.6423
GRASP-ARP-MaxIter=10	4.4375	94.7696
GRASP-ARP-MaxIter=100	42.3155	755.2196

TABLE IV
RELAY PLACEMENT ALGORITHMS' RUN TIME FOR THE MULTIPLE SOURCES - MULTIPLE SINKS IN 49-NODE NETWORKS

Algorithms	**Run time (sec)**	
	$k=2$	$k=3$
K-CONN-REPAIR	283.5649	665.4548
GRASP-ARP-AnySinks-DP	30.1625	274.1320
GRASP-ARP-AnySinks	38.0461	387.7298
GRASP-ARP-DiffSinks-DP	23.8539	336.0117
GRASP-ARP-DiffSinks	42.2258	426.7554

A. Multiple Sources - Single Sink Problem

We choose the sink at the top-left or in the centre of the network, while all sensor nodes are the source nodes. Our simulation uses 25-node topologies and 100 candidate relays. We also set the maximum path length l_{max} equals to 10 for topologies which consist of 25 sensor nodes. GRASP-ARP uses the dynamic programming implementation (DP) and the basic Counting-Paths algorithm to find the k node-disjoint paths. Figure 1 shows the number of additional relay nodes needed for $k = 2$ and $k = 3$ for the case where the sink location is at the top-left of the network, while Figure 2 shows the results for the case where the sink is in the centre. Our GRASP-ARP finds nearly the same number of additional relay nodes compared to K-CONN-REPAIR when the position of the sink is in the centre of the network because it has higher connectivity. These two figures also show that the effect of randomness in GRASP does not guarantee that a higher number of maximum iteration (*MaxIter*) in the local search produces better results. Table II and III shows the algorithms' run time, where for small networks, GRASP-ARP takes longer compared to K-CONN-REPAIR especially for $k = 3$. This happens because GRASP-ARP repeatedly executes Counting-Paths during the local search phase.

B. Multiple Sources - Multiple Sinks Problem

We have four sinks deployed at the top-left, top-right, bottom-left and bottom-right of the network. We use 49-node topologies with 196 candidate relays and set l_{max} equals to 15. In multiple sinks problem, there are two cases where the node-disjoint paths terminate at: *different sinks* and *any sinks*. The different sinks problem is where the k node-disjoint paths must terminate at k different sinks. The any sinks problem is the case where the k node-disjoint paths may terminate at any sinks. Figure 3 and Table IV show the simulation results, where we use *max_iteration* = 10 for GRASP-ARP. In this scenario, GRASP-ARP outperforms K-CONN-REPAIR in both the number of additional relay nodes needed and the run time because GRASP-ARP only finds k node-disjoint paths to the dedicated sinks, while K-CONN-REPAIR must run an expensive connectivity checking algorithm in each iteration to provide k-connectivity for an entire network.

We also vary the number of candidate relays in the networks to show whether it has an impact on the run time efficiency of GRASP-ARP. As depicted in Figure 4 for the 49-node networks, the run time increases significantly when the number of candidate relays increases.

K-CONN-REPAIR is too costly to be implemented in a larger network as it needs to check the connectivity between every pair of nodes. Our GRASP-ARP, on the other hand, is able to scale larger networks. We present the run time for 100 and 225-node topologies in Table V. The results presented here are for $k=2$ and use 400 candidate relays, where 100 and 225-node networks need 0.7 and 1.45 additional relay nodes in average, respectively. More simulation results are available in [14].

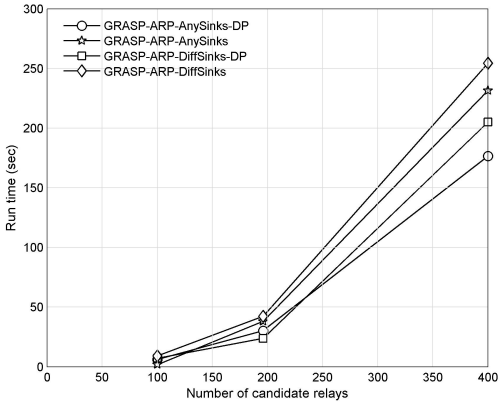

Fig. 4. Algorithms' run time in increasing number of candidate relays for the multiple sources - multiple sinks in 49-node networks

TABLE V
RELAY PLACEMENT ALGORITHMS' RUN TIME FOR THE MULTIPLE SOURCES - MULTIPLE SINKS IN 100 AND 225-NODE NETWORKS

Algorithms	**Run time (sec)**	
	100-node	**225-node**
GRASP-ARP-AnySinks-DP	269.2118	834.9906
GRASP-ARP-AnySinks	307.9353	1358.7140
GRASP-ARP-DiffSinks-DP	261.9937	1027.2200
GRASP-ARP-DiffSinks	305.1702	1245.4740

VI. CONCLUSION AND FUTURE WORK

Ensuring that wireless sensor networks are robust to failures requires that the physical network topology will offer alternative routes to the sink. This requires sensor network deployment to be planned with an objective of ensuring some measure of robustness in the topology, so that when failures do occur that routing protocols can continue to offer reliable delivery. Our contribution is a solution that enables fault-tolerant WSN deployment planning by judicious use of additional relay nodes. In this paper, we define the problem for increasing WSN reliability by deploying a number of additional relay nodes to ensure that each sensor node in the initial design has k node-disjoint paths to the sinks. We present GRASP-ARP, a centralised offline algorithm to be run during the initial topology design to solve this problem. We also adapt a version of the closest approach from the literature for comparison. Our simulation results show that our solution requires fewer relay nodes for larger problems than the competitor, and that different variants of our algorithm are significantly faster, allowing us to tackle larger problems.

We are currently evaluating network protocols and operations using the designed topologies in the network simulator ns-2. While this paper focuses on additional relay node deployment, our future work will include additional sensor deployment, as well as finding locations to deploy sinks in a hop-constrained network that can optimise the network resiliency. We will also consider the networks capacity requirement in the algorithm design.

ACKNOWLEDGMENT

This research is fully funded by the NEMBES project, supported by the Irish Higher Education Authority PRTLI-IV research program.

REFERENCES

[1] D. Torrieri, "Algorithms for Finding an Optimal Set of Short Disjoint Paths in a Communication Network," *IEEE Transactions on Communications*, vol. 40, no. 11, pp. 1698–1702, 1992.

[2] T. Feo and M. Resende, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[3] B. Hao, J. Tang, and G. Xue, "Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Formulation and Approximation," in *Proc. Workshop High Performance Switching and Routing (HPSR'04)*, 2004, pp. 246–250.

[4] J. Bredin, E. Demaine, M. Hajiaghayi, and D. Rus, "Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance," in *Proc. 6th ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc'05)*, 2005, pp. 309–319.

[5] J. Pu, Z. Xiong, and X. Lu, "Fault-Tolerant Deployment with k-connectivity and Partial k-connectivity in Sensor Networks," *Wireless Communications and Mobile Computing*, vol. 9, no. 7, pp. 909–919, 2008.

[6] A. Kashyap, S. Khuller, and M. Shayman, "Relay Placement for Fault Tolerance in Wireless Networks in Higher Dimensions," *Computational Geometry*, 2010.

[7] X. Han, X. Cao, E. Lloyd, and C. Shen, "Fault-tolerant Relay Node Placement in Heterogeneous Wireless Sensor Networks," *IEEE Trans. Mobile Computing*, vol. 9, no. 5, pp. 643–656, 2010.

[8] W. Zhang, G. Xue, and S. Misra, "Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Problems and Algorithms," in *Proc. 26th Ann. IEEE Conf. Computer Communications (INFOCOM'07)*, 2007, pp. 1649–1657.

[9] S. Misra, S. Hong, G. Xue, and J. Tang, "Constrained Relay Node Placement in Wireless Sensor Networks to Meet Connectivity and Survivability Requirements," in *Proc. 27th Ann. IEEE Conf. Computer Communications (INFOCOM'08)*, 2008, pp. 281–285.

[10] R. Bhandari, "Optimal Physical Diversity Algorithms and Survivable Networks," in *Proc. 2nd IEEE Symp. Computers and Communications (ISCC'97)*, 1997, pp. 433–441.

[11] L. Ford and D. Fulkerson, *Flows in Networks*. Princeton, University Press, 1962.

[12] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley, 2011.

[13] S. Martins, P. Pardalos, M. Resende, and C. Ribeiro, "Greedy Randomized Adaptive Search Procedures for the Steiner Problem in Graphs," in *Randomization Methods in Algorithmic Design*, ser. DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1999, vol. 43, pp. 133–145.

[14] L. Sitanayah, K. Brown, and C. Sreenan, "Relay Node Placement for k Node-Disjoint Paths in Constrained Wireless Sensor Network Deployment Planning," Dept. of Computer Science, University College Cork, Tech. Rep. UCC-CS-2011-07-17, Jul. 2011.

[15] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice Hall, 1982.

[16] R. Ravi and D. Williamson, "An Approximation Algorithm for Minimum-Cost Vertex-Connectivity Problems," *Algorithmica*, vol. 18, no. 1, pp. 21–43, 1997.